DEFINIÇÃO

Junções interior e exterior. Consultas correlatas e aninhadas. Operadores de conjunto.

PROPÓSITO

Projetar consultas envolvendo diversas tabelas com diferentes tipos de junção e a utilização de mecanismos de subconsultas aninhadas, correlatas ou mesmo com o uso de operadores de conjunto.

PREPARAÇÃO

Antes de iniciar o conteúdo deste tema, certifique-se de ter baixado e instalado o SGBD PostgreSQL em seu computador.

OBJETIVOS

MÓDULO 1

Operar consultas envolvendo junções interior e exterior

MÓDULO 2

Operar subconsultas aninhadas e correlatas

MÓDULO 3

Operar consultas com o uso de operadores de conjunto

INTRODUÇÃO

Ao longo deste tema, aprenderemos a executar consultas envolvendo mais de uma tabela. Quando projetamos um banco de dados relacional, em geral, o esquema final possuirá diversas tabelas relacionadas.

Além disso, algumas tabelas são mais independentes do que outras. Na prática, podemos afirmar que uma tabela que não contenha restrição de chave estrangeira em alguma coluna é mais independente do que a(s) que a tem.

Assim, é comum a necessidade de projetarmos consultas que recuperem dados de diferentes tabelas. Vamos perceber que será fundamental entendermos de que maneira se dá o provável relacionamento entre as tabelas envolvidas.

Aprenderemos também que, na maior parte dos casos, o entendimento sobre o relacionamento entre tabelas ficará mais explícito quando localizarmos as colunas envolvidas na relação entre elas: chave primária de uma tabela e a respectiva chave estrangeira em outra ou na mesma tabela (no caso de autorrelacionamento).

Clique aqui e baixe o arquivo com os códigos das consultas utilizados nos exemplos apresentados nos módulos.

MÓDULO 1

- Operar consultas envolvendo junções interior e exterior

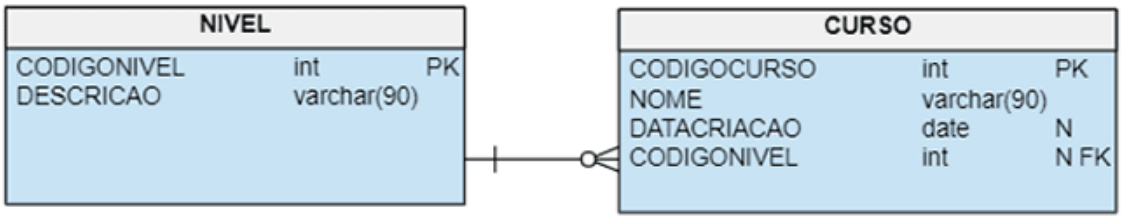
OPERAÇÃO DE JUNÇÃO DE TABELAS

A JUNÇÃO DE TABELAS É UMA DAS OPERAÇÕES MAIS IMPORTANTES CRIADAS PELO MODELO RELACIONAL DE BANCO DADOS. NA DEFINIÇÃO MATEMÁTICA, O RESULTADO DA JUNÇÃO DE DUAS TABELAS É UM SUBCONJUNTO DO PRODUTO CARTESIANO ENTRE ESSAS TABELAS. A ESPECIFICAÇÃO DESSE SUBCONJUNTO É FEITA POR UMA CONDIÇÃO DE JUNÇÃO ENTRE COLUNAS DAS DUAS TABELAS.

Veremos que existem dois tipos de junção: interna, denominada INNER JOIN; e externa, denominada OUTER JOIN que, por sua vez, pode ser LEFT, FULL ou RIGHT OUTER JOIN.

EXEMPLOS DE TABELAS

Construiremos algumas consultas com base nas tabelas NIVEL e CURSO, conforme figura a seguir:



Fonte: O autor.

Tabelas NIVEL e CURSO.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessando algum *database* criado por você.

```
1  CREATE TABLE NIVEL (
2    CODIGONIVEL int  NOT NULL,
3    DESCRICAO varchar(90)  NOT NULL,
4    CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL));
5
6 CREATE TABLE CURSO (
7    CODIGOCURSO int  NOT NULL,
8    NOME varchar(90)  NOT NULL UNIQUE,
9    DATACRIACAO date  NULL,
10   CODIGONIVEL int  NULL,
11   CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO));
12 ALTER TABLE CURSO ADD FOREIGN KEY (CODIGONIVEL) REFERENCES NIVEL;
13
14 INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (1,'Graduação');
15 INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (2,'Especialização');
16 INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (3,'Mestrado');
17 INSERT INTO NIVEL (CODIGONIVEL, DESCRICAO) VALUES (4,'Doutorado');
18 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (1,'Sistemas de Informação','19/06/1999',1);
19 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (2,'Medicina','10/05/1990',1);
20 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (3,'Nutrição','19/02/2012',NULL);
21 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (4,'Pedagogia','19/06/1999',1);
22 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (5,'Saúde da Família','10/09/1999',3);
23 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO,CODIGONIVEL) VALUES (6,'Computação Aplicada','10/09/1999',NULL);
```

Fonte: O autor.

No *script*, foram usados três comandos: CREATE, ALTER e INSERT. A sintaxe completa desses comandos no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore + ao final do tema.

A figura a seguir apresenta o conteúdo da tabela NIVEL após a execução do comando SELECT * FROM NIVEL;

	codigonivel	descricao
1	1	Graduação
2	2	Especialização
3	3	Mestrado
4	4	Doutorado

Fonte: O autor.

Conteúdo da tabela NIVEL.

A figura a seguir apresenta o conteúdo da tabela CURSO após a execução do comando TABLE CURSO;

	codigocurso	nome	datacriacao	codigonivel
1	1	Sistemas de Informação	1999-06-19	1
2	2	Medicina	1990-05-10	1
3	3	Nutrição	2012-02-19	[NULL]
4	4	Pedagogia	1999-06-19	1
5	5	Saúde da Família	1999-09-10	3
6	6	Computação Aplicada	1999-09-10	[NULL]

Fonte: O autor.

Conteúdo da tabela CURSO.

Observe que as tabelas NIVEL e CURSO estão relacionadas. A tabela NIVEL é a mais independente, visto que não possui chave estrangeira. Consequência disso é a possibilidade de um usuário cadastrar diversos níveis na tabela, sem a preocupação com qualquer outra tabela do banco de dados. O relacionamento entre NIVEL e CURSO está implementado na tabela CURSO, por meio da coluna CODIGONIVEL, que exerce o papel de chave estrangeira.

Por fim, note também que o fato da coluna CODIGONIVEL ser opcional na tabela CURSO representa a possibilidade de cadastrar um curso sem, em um primeiro momento, relacioná-lo a determinado nível existente. É justamente o que ocorreu após a execução dos comandos das linhas 20 e 23, nos quais o valor inserido para a coluna CODIGONIVEL é NULL.

A seguir, algumas observações sobre o que acontece quando declaramos mais de uma tabela em uma consulta SQL.

OPERAÇÃO DE PRODUTO CARTESIANO

Em termos estruturais, a tabela NIVEL possui duas colunas e quatro registros. De forma semelhante, a tabela CURSO possui quatro colunas e seis registros. Você pode chegar a essa

conclusão ao analisar o *script* anterior.

Vamos, agora, executar a seguinte consulta SQL: SELECT * FROM CURSO, NIVEL; cujo resultado está expresso na figura a seguir:

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	1	Sistemas de Informação	1999-06-19	1	2	Especialização
3	1	Sistemas de Informação	1999-06-19	1	3	Mestrado
4	1	Sistemas de Informação	1999-06-19	1	4	Doutorado
5	2	Medicina	1990-05-10	1	1	Graduação
6	2	Medicina	1990-05-10	1	2	Especialização
7	2	Medicina	1990-05-10	1	3	Mestrado
8	2	Medicina	1990-05-10	1	4	Doutorado
9	3	Nutrição	2012-02-19	[NULL]	1	Graduação
10	3	Nutrição	2012-02-19	[NULL]	2	Especialização
11	3	Nutrição	2012-02-19	[NULL]	3	Mestrado
12	3	Nutrição	2012-02-19	[NULL]	4	Doutorado
13	4	Pedagogia	1999-06-19	1	1	Graduação
14	4	Pedagogia	1999-06-19	1	2	Especialização
15	4	Pedagogia	1999-06-19	1	3	Mestrado
16	4	Pedagogia	1999-06-19	1	4	Doutorado
17	5	Saúde da Família	1999-09-10	3	1	Graduação
18	5	Saúde da Família	1999-09-10	3	2	Especialização
19	5	Saúde da Família	1999-09-10	3	3	Mestrado
20	5	Saúde da Família	1999-09-10	3	4	Doutorado
21	6	Computação Aplicada	1999-09-10	[NULL]	1	Graduação
22	6	Computação Aplicada	1999-09-10	[NULL]	2	Especialização
23	6	Computação Aplicada	1999-09-10	[NULL]	3	Mestrado
24	6	Computação Aplicada	1999-09-10	[NULL]	4	Doutorado

Fonte: O autor.

📷 Resultado da consulta SELECT * FROM CURSO, NIVEL;

O resultado da consulta anterior é uma tabela - em memória principal - originada da operação de produto cartesiano. Nessa operação, o sistema gerenciador de banco de dados (SGBD) combinou cada linha da tabela CURSO com cada registro da tabela NIVEL.

Note que a quantidade de (seis) colunas na tabela resultante é a soma das colunas das tabelas envolvidas. De forma semelhante, a quantidade (vinte e quatro) de registros da tabela é igual ao produto entre o número de linhas de CURSO e NIVEL.

Perceba que, ao longo do nosso estudo, temos interpretado cada linha de uma tabela como sendo um fato registrado no banco de dados, correspondendo a uma realidade do contexto do negócio sendo modelado.

Exemplo: ao visualizarmos o conteúdo da tabela NIVEL, afirmamos que há quatro registros ou ocorrências de nível no banco de dados. De forma semelhante, há seis cursos cadastrados na tabela CURSO.

No entanto, o mesmo raciocínio não pode ser aplicado à tabela resultante da consulta, já que temos todas as combinações possíveis entre as linhas, além de envolver todas as colunas das tabelas CURSO e NIVEL.

COMENTÁRIO

Cabe aqui um alerta sobre o cuidado que se deve ter ao usar esse tipo de consulta, que resulta em produto cartesiano. Suponha uma tabela CLIENTES com 1.000.000 de linhas e uma tabela PEDIDOS com 10.000.000 de linhas. O comando SELECT * FROM CLIENTES, PEDIDOS retornaria 10.000.000.000.000 de linhas e, provavelmente, ocuparia longas horas de processamento no servidor, além de retornar um resultado inútil!

Há outra maneira de obtermos os mesmos resultados de SELECT * FROM CURSO, NIVEL;?

Sim. Basta executar o código equivalente: SELECT * FROM CURSO CROSS JOIN NIVEL;

A seguir, vamos estudar uma forma de extrair informações úteis a partir do resultado dessa consulta.

JUNÇÃO INTERNA

Estamos interessados em obter informações úteis para o nosso usuário. Perceba que, por exemplo, o curso de Sistemas de Informação está classificado como um curso pertencente ao nível de graduação.

Como chegamos a essa conclusão? Avalie o conteúdo das linhas 14 e 18 do *script* que insere informações nas tabelas NIVEL e CURSO.

Agora, examine as quatro primeiras linhas da tabela resultante da consulta anterior, expressa na figura a seguir:

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	1	Sistemas de Informação	1999-06-19	1	2	Especialização
3	1	Sistemas de Informação	1999-06-19	1	3	Mestrado
4	1	Sistemas de Informação	1999-06-19	1	4	Doutorado

Fonte: O autor.

Subconjunto da tabela resultante do produto cartesiano entre CURSO e NIVEL.

Note que, se examinarmos cada linha como um fato, vamos perceber que somente a primeira corresponde à realidade cadastrada no banco de dados. Não por coincidência, perceba que os valores das colunas CODIGONIVEL são os mesmos. Nas três últimas, diferentes.

Assim, nós podemos eliminar as linhas falsas se programarmos uma condição de igualdade envolvendo as colunas em questão. Devemos lembrar que as quatro primeiras colunas vêm da tabela CURSO. As duas últimas são originadas na tabela NIVEL.

Para recuperar as linhas que correspondem à realidade cadastrada no banco de dados, você pode executar o comando a seguir:

```
1 SELECT *
2 FROM CURSO, NIVEL
3 WHERE NIVEL.CODIGONIVEL=CURSO.CODIGONIVEL;
```

O resultado dessa consulta está expresso na figura a seguir:

	codigocurso	nome	datacriacao	codigonivel	codigonivel	descricao
1	1	Sistemas de Informação	1999-06-19	1	1	Graduação
2	2	Medicina	1990-05-10	1	1	Graduação
3	4	Pedagogia	1999-06-19	1	1	Graduação
4	5	Saúde da Família	1999-09-10	3	3	Mestrado

Fonte: O autor.

Resultado da consulta envolvendo as tabelas CURSO e NIVEL.

Perceba que foram recuperadas as linhas que de fato relacionam cursos aos seus respectivos níveis. No entanto, a forma mais usada para retornar os mesmos resultados é com o auxílio da cláusula de junção interna, o qual possui sintaxe **básica** conforme a seguir:

SELECT *
FROM TABELA1 [INNER] JOIN TABELA2 ON (CONDIÇÃOJUNÇÃO) [USING (COLUNA_DE_JUNÇÃO)]

Na sintaxe apresentada, declaramos a cláusula de junção entre as tabelas. Em seguida, a preposição “ON” e a condição da junção. Na maioria das vezes, essa condição corresponderá a uma igualdade envolvendo a chave primária de uma tabela e a chave estrangeira correspondente. A sintaxe completa do comando SELECT no PostgreSQL pode ser encontrada com a indicação disponibilizada no Explore + ao final do tema.

Veja, a seguir, o código SQL correspondente ao nosso exemplo:

```
1 SELECT *
2 FROM CURSO INNER JOIN NIVEL ON(NIVEL.CODIGONIVEL=CURSO.CODIGONIVEL);
```

Note também que é possível declarar a cláusula USING especificando a coluna alvo da junção. No caso do nosso exemplo, a coluna CODIGONIVEL. A consulta pode ser reescrita conforme a seguir:

```
1 SELECT *
2 FROM CURSO INNER JOIN NIVEL USING(CODIGONIVEL);
```

Se desejarmos exibir o código e o nome do curso, além do código e o nome do nível, podemos, então executar o código a seguir:

```
1 SELECT CURSO.CODIGOCURSO, CURSO.NOME,
2 FROM CURSO INNER JOIN NIVEL USING(CODIGONIVEL);
```

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3FROM CURSO INNER JOIN NIVEL ON
4 (NIVEL.CODIGONIVEL=CURSO.CODIGONIVEL;
```

Perceba que, no comando SELECT, usamos uma referência mais completa: NOMETABELA.NOMECOLUNA. Essa referência só é obrigatória para a coluna CODIGONIVEL, uma vez que é necessário especificar de qual tabela o SGBD irá buscar os valores. No entanto, em termos de organização de código, é interessante usar esse tipo de referência para cada coluna.

Finalmente, observe que poderíamos também, no contexto da consulta, renomear as tabelas envolvidas, deixando o código mais elegante e legível, conforme a seguir:

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3 FROM CURSO C INNER JOIN NIVEL N ON
4 (N.CODIGONIVEL=C.CODIGONIVEL) ;
```

O resultado da consulta está expresso na figura a seguir:

	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	4	Pedagogia	1	Graduação
4	5	Saúde da Família	3	Mestrado

Fonte: O autor.

Resultado da consulta envolvendo as tabelas CURSO e NIVEL.

O resultado de uma junção interna corresponde somente aos registros que atendem à condição da junção, ou seja, os registros que de fato estão relacionados no contexto das tabelas envolvidas.

JUNÇÃO EXTERNA

O resultado da consulta anterior exibe somente os cursos para os quais há registro de informação sobre o nível associado a eles. E se quiséssemos incluir na listagem todos os registros da tabela CURSO?

Para incluir no resultado da consulta todas as ocorrências da tabela CURSO, podemos usar a cláusula LEFT JOIN (junção à esquerda). Nesse tipo de junção, o resultado contém todos os registros da tabela declarada à esquerda da cláusula JOIN, mesmo que não haja registros correspondentes na tabela da direita. Em especial, quando não há correspondência, os resultados são retornados com o valor NULL.

Veja a seguir exemplo de uso de junção à esquerda:

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3FROM CURSO C LEFT JOIN NIVEL N ON (N.CODIGONIVEL=C.CODIGONIVEL) ;
```

A figura a seguir apresenta o resultado da consulta anterior:

	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	3	Nutrição	[NULL]	[NULL]
4	4	Pedagogia	1	Graduação
5	5	Saúde da Família	3	Mestrado
6	6	Computação Aplicada	[NULL]	[NULL]

Fonte: O autor.

Cursos e níveis, exibindo todos os registros da tabela CURSO.

Observe que o número de linhas do resultado da consulta coincide com o número de linhas da tabela CURSO, visto que todos os registros dessa tabela fazem parte do resultado e a chave estrangeira que implementa o relacionamento está localizada na tabela CURSO. Em especial, as linhas 3 e 6 correspondem a cursos onde não há informação sobre o nível associado a eles.

Perceba também que, de forma semelhante, poderíamos ter interesse em exibir todos os registros da tabela à direita da cláusula JOIN. Em nosso exemplo, a tabela NIVEL. A cláusula RIGHT JOIN (junção à direita) é usada para essa finalidade. Nesse tipo de junção, o resultado contém todos os registros da tabela declarada à direita da cláusula JOIN, mesmo que não haja registros correspondentes na tabela da esquerda. Em especial, quando não há correspondência, os resultados são retornados com o valor NULL.

Veja a seguir exemplo de uso de junção à direita:

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3FROM CURSO C RIGHT JOIN NIVEL N ON
4(N.CODIGONIVEL=C.CODIGONIVEL);
```

A figura a seguir apresenta o resultado da consulta anterior:

	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	4	Pedagogia	1	Graduação
4	5	Saúde da Família	3	Mestrado
5	[NULL]	[NULL]	2	Especialização
6	[NULL]	[NULL]	4	Doutorado

Fonte: O autor.

Cursos e níveis, exibindo todos os registros da tabela NIVEL.

Note que todos os registros da tabela NIVEL aparecem no resultado. As quatro primeiras linhas do resultado da consulta correspondem aos registros que efetivamente estão relacionados à tabela CURSO. As duas últimas linhas são registros que não estão relacionados a qualquer curso existente no banco de dados.

Perceba que o mesmo resultado pode ser obtido se usarmos junção à esquerda e junção à direita, *alternando* a posição das tabelas envolvidas. Com isso, queremos dizer que TABELA1 LEFT JOIN TABELA2 é equivalente a TABELA2 RIGHT JOIN TABELA1.

Veja exemplo a seguir:

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3 FROM CURSO C LEFT JOIN NIVEL N ON (N.CODIGONIVEL=C.CODIGONIVEL);
```

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3 FROM NIVEL N RIGHT JOIN CURSO C ON (N.CODIGONIVEL=C.CODIGONIVEL);
```

Outro tipo de junção externa, denominada FULL OUTER JOIN (junção completa), apresenta todos os registros das tabelas à esquerda e à direita, mesmo os registros não relacionados. Em outras palavras, a tabela resultante exibirá todos os registros de ambas as tabelas, além de valores NULL no caso dos registros sem correspondência. Veja exemplo a seguir:

```
1 SELECT C.CODIGOCURSO, C.NOME,
2 N.CODIGONIVEL, N.DESCRICAO
3 FROM CURSO C FULL OUTER JOIN NIVEL N ON
4 (N.CODIGONIVEL=C.CODIGONIVEL);
```

A figura a seguir apresenta o resultado da consulta anterior:

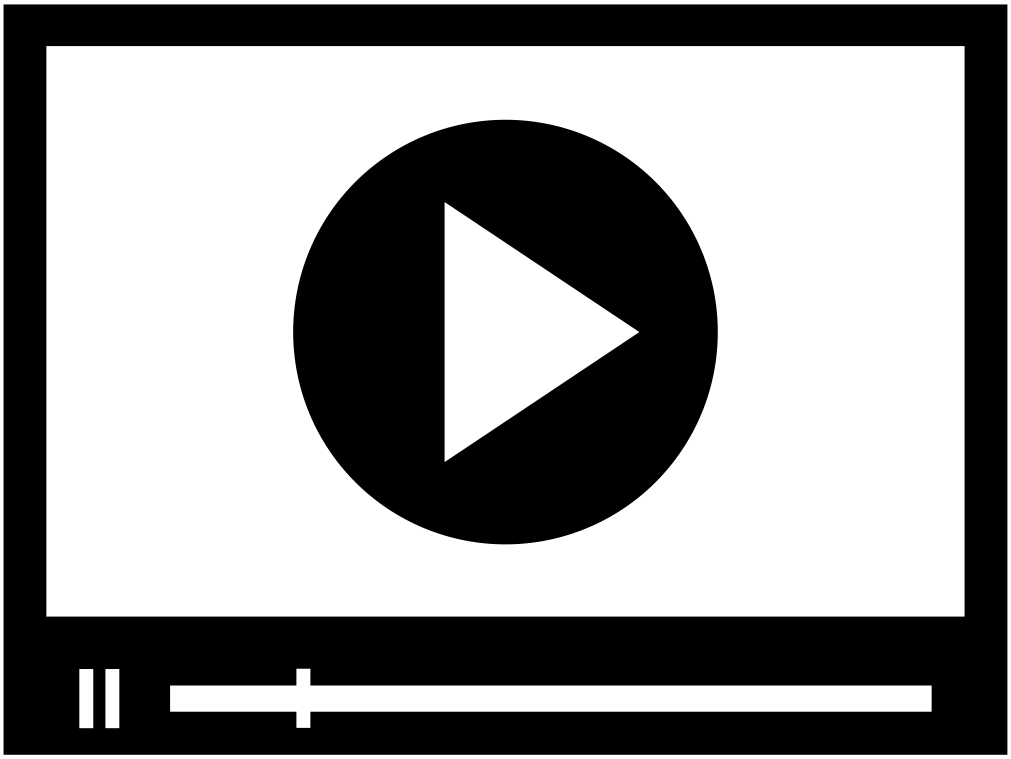
	codigocurso	nome	codigonivel	descricao
1	1	Sistemas de Informação	1	Graduação
2	2	Medicina	1	Graduação
3	3	Nutrição	[NULL]	[NULL]
4	4	Pedagogia	1	Graduação
5	5	Saúde da Família	3	Mestrado
6	6	Computação Aplicada	[NULL]	[NULL]
7	[NULL]	[NULL]	2	Especialização
8	[NULL]	[NULL]	4	Doutorado

Fonte: O autor.

Resultado do FULL OUTER JOIN entre CURSO e NIVEL.

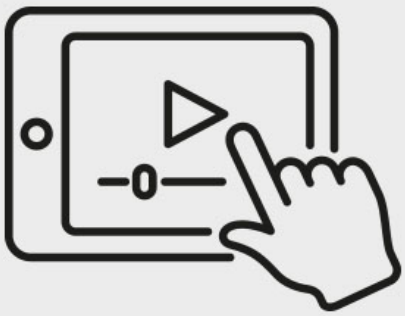
Note que, no resultado, aparecem os registros de cada tabela. Além disso, valores NULL são exibidos nos casos em que não há correspondência entre as tabelas (linhas 3, 6, 7 e 8).

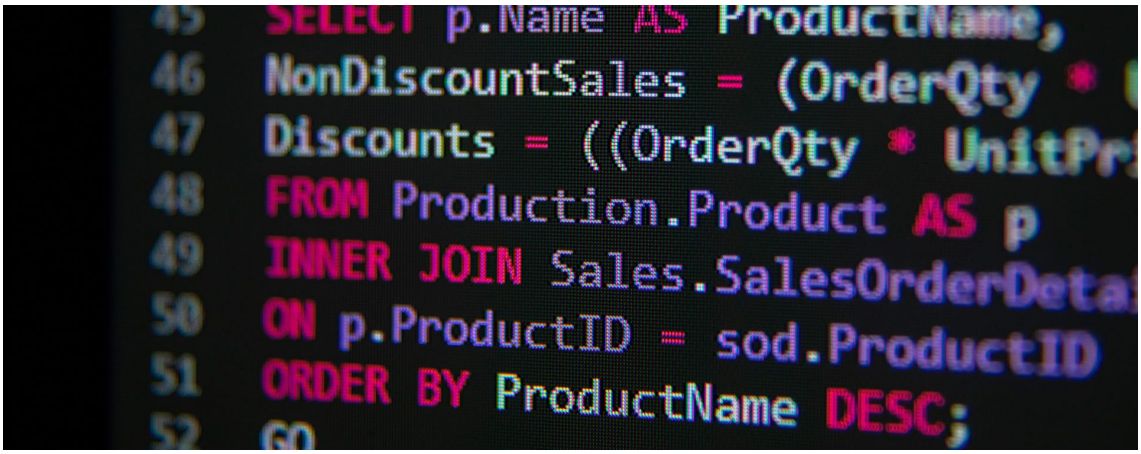
Neste módulo, aprendemos que, quando há necessidade de extrair informações de mais de uma tabela, utilizamos a cláusula de junção. Estudamos diversos tipos de junção, que serão utilizados de acordo com a necessidade do usuário, visto que cada tipo de junção possui uma especificidade.



CONSULTAS NO POSTGRESQL SOBRE AS TABELAS NÍVEL E CURSO

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

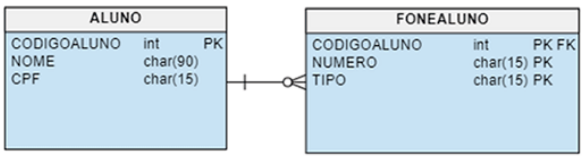




Fonte: Shutterstock.

VERIFICANDO O APRENDIZADO

1. CONSIDERE AS TABELAS E O CÓDIGO SQL A SEGUIR:




```
1 SELECT NOME, NUMERO, TIPO
2 FROM ALUNO A LEFT JOIN FONEALUNO F
3     ON (A.CODIGOALUNO=F.CODIGOALUNO);
```

A CONSULTA RETORNA:

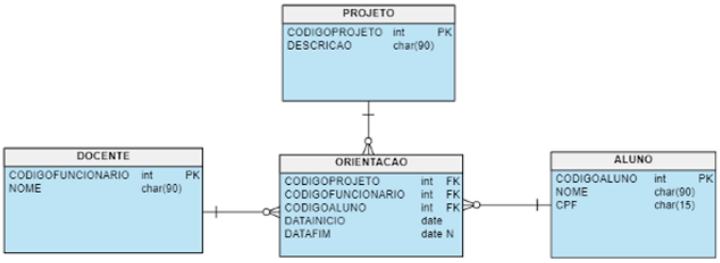
A) Somente os registros de ALUNO relacionados à tabela FONEALUNO.

B) Registros, tanto dos alunos com telefone, quanto dos alunos sem telefone.

C) Erro.

D) Somente registros dos alunos que têm mais de um telefone cadastrado.

2. CONSIDERE AS TABELAS E O CÓDIGO SQL A SEGUIR:




```
1 SELECT PR.DESCRICAO,D.NOME, A.NOME
2 FROM ALUNO A JOIN ORIENTACAO O ON (A.CODIGOALUNO=O.CODIGOALUNO)
3      JOIN DOCENTE D ON (D.CODIGOFUNCIONARIO=O.CODIGOFUNCIONARIO)
4      JOIN PROJETO PR ON (PR.CODIGOPROJETO=O.CODIGOPROJETO)
5 ORDER BY PR.DESCRICAO,D.NOME, A.NOME;
```

A CONSULTA RETORNA:

A) A descrição dos projetos e nomes de professores e alunos envolvidos em orientação.

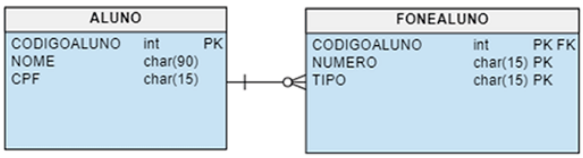
B) A descrição dos projetos e nomes de professores e alunos não envolvidos em orientação.

C) A descrição dos projetos e nomes de professores e alunos envolvidos e não envolvidos em orientação.

D) A descrição dos projetos sem orientação definida.

GABARITO

1. Considere as tabelas e o código SQL a seguir:



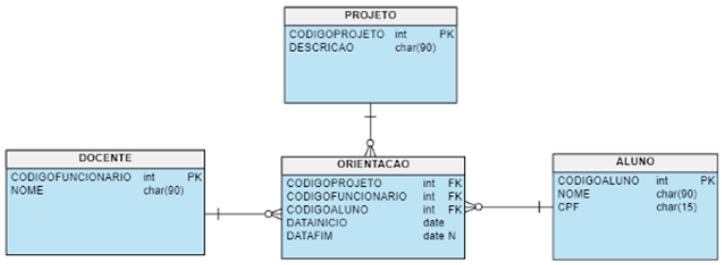

```
1 SELECT NOME, NUMERO, TIPO
2 FROM ALUNO A LEFT JOIN FONEALUNO F
3     ON (A.CODIGOALUNO=F.CODIGOALUNO);
```

A consulta retorna:

A alternativa **"B "** está correta.

Parabéns! A alternativa B está correta. A consulta usa junção à esquerda. Logo, todos os registros de ALUNO farão parte do resultado: alunos com e sem telefone cadastrado no banco de dados.

2. Considere as tabelas e o código SQL a seguir:




```
1 SELECT PR.DESCRICAO,D.NOME, A.NOME
2 FROM ALUNO A JOIN ORIENTACAO O ON (A.CODIGOALUNO=O.CODIGOALUNO)
3      JOIN DOCENTE D ON (D.CODIGOFUNCIONARIO=O.CODIGOFUNCIONARIO)
4      JOIN PROJETO PR ON (PR.CODIGOPROJETO=O.CODIGOPROJETO)
5 ORDER BY PR.DESCRICAO,D.NOME, A.NOME;
```

A consulta retorna:

A alternativa **"A "** está correta.

A consulta envolve somente junções internas. Retorna informações sobre todas as orientações cadastradas no banco de dados.

MÓDULO 2

⦿ Operar subconsultas aninhadas e correlatas

SUBCONSULTAS

Podemos admitir que o resultado de uma consulta SQL corresponde a uma tabela, mesmo que esteja temporariamente armazenada na memória principal do computador.

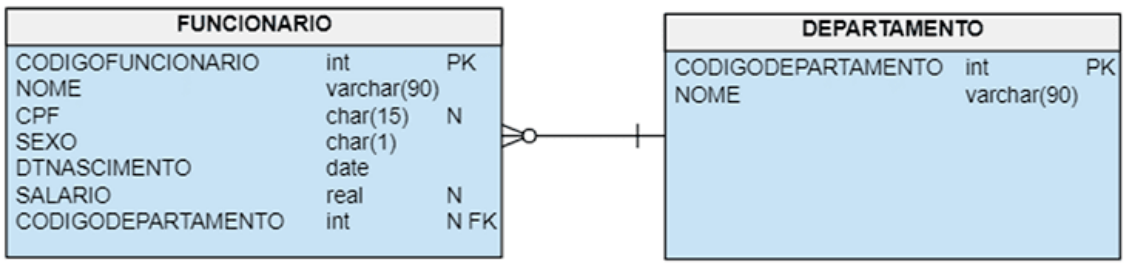
Vamos construir consultas que dependem dos - ou usam - resultados de outras consultas para recuperar informações de interesse. Essa categoria é conhecida por subconsulta, uma consulta sobre o resultado de outra consulta.

Ao longo do nosso estudo, vamos explorar dois tipos de subconsultas: **aninhadas e correlatas**. No primeiro caso, a consulta mais externa realizará algum tipo de teste junto aos dados originados da consulta mais interna. No segundo, a subconsulta utiliza valores da consulta externa. Nesse tipo de consulta, a subconsulta é executada para cada linha da consulta externa.

Exemplo de tabelas

Construiremos as consultas com base nas tabelas FUNCIONARIO E DEPARTAMENTO, conforme figura a seguir.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessar algum *database* criado por você.



Fonte: O autor.

Tabelas FUNCIONARIO e DEPARTAMENTO.

```sql
CREATE TABLE DEPARTAMENTO (
   CODIGODEPARTAMENTO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CONSTRAINT DEPARTAMENTO_pk PRIMARY KEY (CODIGODEPARTAMENTO));
CREATE TABLE FUNCIONARIO (
   CODIGOFUNCIONARIO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CPF char(15)  NULL,
   SEXO char(1)  NOT NULL,
   DTNASCIMENTO date  NOT NULL,
   SALARIO real   NULL,
   CODIGODEPARTAMENTO int  NULL,
   CONSTRAINT FUNCIONARIO_pk PRIMARY KEY (CODIGOFUNCIONARIO));
ALTER TABLE FUNCIONARIO ADD CONSTRAINT FUNCIONARIO_DEPARTAMENTO
   FOREIGN KEY (CODIGODEPARTAMENTO)  REFERENCES DEPARTAMENTO (CODIGODEPARTAMENTO) ;

INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (1,'Tecnologia da Informação');
INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (2,'Contabilidade');
INSERT INTO DEPARTAMENTO(CODIGODEPARTAMENTO,NOME) VALUES (3,'Marketing');
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,CODIGODEPARTAMENTO)
VALUES (1,'ROBERTA SILVA BRASIL',NULL,'F','20/02/1980',7000,1);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,CODIGODEPARTAMENTO)
VALUES (2,'MARIA SILVA BRASIL',NULL,'F','20/09/1988',9500,2);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,CODIGODEPARTAMENTO)
VALUES (3,'GABRIELLA PEREIRA LIMA',NULL,'F','20/02/1990',6000,1);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,CODIGODEPARTAMENTO)
VALUES (4,'MARCOS PEREIRA BRASIL',NULL,'M','20/02/1999',6000,2);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO,CODIGODEPARTAMENTO)
VALUES (5,'HEMERSON SILVA BRASIL',NULL,'M','20/12/1992',4000,NULL);
```

Fonte: O autor.

A figura a seguir apresenta o conteúdo da tabela DEPARTAMENTO após a execução do comando

SELECT * FROM DEPARTAMENTO;

	codigodepartamento	nome
1	1	Tecnologia da Informação
2	2	Contabilidade
3	3	Marketing

Fonte: O autor.

📷 Conteúdo da tabela DEPARTAMENTO.

A figura a seguir apresenta o conteúdo da tabela FUNCIONARIO após a execução do comando

TABLE FUNCIONARIO; (equivalente a SELECT * FROM FUNCIONARIO;)

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
3	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	6.000	1
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	6.000	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Fonte: O autor.

📷 Conteúdo da tabela FUNCIONARIO.

Perceba que, originalmente, existe um relacionamento do tipo 1:N entre DEPARTAMENTO e FUNCIONARIO. A implementação desse relacionamento ocorre por meio da chave estrangeira CODIGODEPARTAMENTO da tabela FUNCIONARIO.

SUBCONSULTAS ANINHADAS

Uma subconsulta aninhada ocorre quando é necessário obter dados que dependem do resultado de uma - ou mais - consulta(s) mais interna(s). Para isso, cria-se uma condição na cláusula WHERE de forma a envolver o resultado da subconsulta em algum tipo de teste.

Vamos estudar alguns exemplos?

Consulta 01: Retornar o código e o nome do(s) funcionário(s) que ganha(m) o maior salário.

Solução:

```
1 SELECT CODIGOFUNCIONARIO, NOME
2 FROM FUNCIONARIO
3 WHERE SALARIO=
4                 (SELECT MAX(SALARIO)
5                 FROM FUNCIONARIO);
```

Fonte: O autor.

O resultado da consulta 01 está expresso na figura a seguir:

	codigofuncionario	nome
1	2	MARIA SILVA BRASIL

Fonte: O autor.

Resultado da consulta 01.

COMENTÁRIO

Inicialmente, o SGBD processa a subconsulta, a qual retorna o valor do maior salário registrado na tabela FUNCIONARIO. Em seguida, esse resultado é utilizado na avaliação da cláusula

WHERE. Finalmente, são exibidos os registros dos funcionários cujo valor de SALARIO satisfaz a condição da cláusula WHERE.

Consulta 02: Retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) mais que a média salarial dos colaboradores.

Solução:

```
1 SELECT CODIGOFUNCIONARIO, NOME, SALARIO
2 FROM FUNCIONARIO
3 WHERE SALARIO>
4                 (SELECT AVG(SALARIO)
5                 FROM FUNCIONARIO);
```

Fonte: O autor.

O resultado da consulta 02 está expresso na figura a seguir:

	codigofuncionario	nome	salario
1	1	ROBERTA SILVA BRASIL	7.000
2	2	MARIA SILVA BRASIL	9.500

Fonte: O autor.

Resultado da consulta 02.

COMENTÁRIO

Inicialmente, o SGBD processa a subconsulta, a qual retorna a média salarial a partir da tabela FUNCIONARIO. Em seguida, esse resultado é utilizado na avaliação da cláusula WHERE. Finalmente, são exibidas as linhas da tabela FUNCIONARIO com as colunas CODIGOFUNCIONARIO, NOME e SALARIO, cujo valor de SALARIO satisfaz a condição da cláusula WHERE.

Consulta 03: Retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) menos que a média salarial dos colaboradores do departamento de Tecnologia da Informação.

Solução:

```
1 SELECT CODIGOFUNCIONARIO, NOME, SALARIO
2 FROM FUNCIONARIO
3 WHERE SALARIO<
4         (SELECT AVG(SALARIO)
5         FROM FUNCIONARIO
6         WHERE CODIGODEPARTAMENTO IN (SELECT CODIGODEPARTAMENTO
7                                      FROM DEPARTAMENTO
8                                      WHERE NOME='Tecnologia da Informação'));
```

Fonte: O autor.

O resultado da consulta 03 está expresso na figura a seguir:

	codigofuncionario	nome	salario
1	3	GABRIELLA PEREIRA LIMA	6.000
2	4	MARCOS PEREIRA BRASIL	6.000
3	5	HEMERSON SILVA BRASIL	4.000

Fonte: O autor.

Resultado da consulta 03.

COMENTÁRIO

Perceba que, para retornar os resultados de interesse, o SGBD precisa calcular a média salarial dos funcionários do departamento de Tecnologia da Informação. Para isso, a subconsulta da linha 4 – que calcula essa média - utiliza o resultado da subconsulta da linha 6, a qual recupera o código do departamento de Tecnologia da Informação).

Há outra maneira de resolver a consulta 03?

Sim, você pode trocar uma subconsulta por uma junção. O código a seguir produz os mesmos resultados:

```
SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO
WHERE SALARIO<
        (SELECT AVG(SALARIO)
        FROM FUNCIONARIO F JOIN DEPARTAMENTO D ON
            (F.CODIGODEPARTAMENTO=D.CODIGODEPARTAMENTO)
        WHERE D.NOME='Tecnologia da Informação');
```

Fonte: O autor.

Consulta 04: Quantos funcionários recebem menos que a funcionária que possui o maior salário entre as colaboradoras de sexo feminino?

Solução:

```
1 SELECT COUNT(*) AS QUANTIDADE
2 FROM FUNCIONARIO
3 WHERE SALARIO<
4         (SELECT MAX(SALARIO)
5         FROM FUNCIONARIO
6         WHERE SEXO='F');
```

Fonte: O autor.

O resultado da consulta 04 está expresso na figura a seguir:

	quantidade
1	4

Fonte: O autor.

Resultado da consulta 04.

COMENTÁRIO

No exemplo, o SGBD recupera o maior salário entre as funcionárias. Em seguida, conta os registros cujo valor da coluna SALARIO é menor que o valor do salário em questão.

SUBCONSULTAS CORRELATAS

Uma subconsulta correlata – ou correlacionada – é um tipo especial de consulta aninhada. Uma consulta correlata ocorre quando é necessário obter dados que dependem do resultado de uma - ou mais - consulta(s) mais interna(s). Para isso, cria-se uma condição na cláusula WHERE de forma a envolver o resultado da subconsulta em algum tipo de teste.

Vamos estudar alguns exemplos?

Consulta 05: Retornar o código, o nome e o salário do(s) funcionário(s) que ganha(m) mais que a média salarial dos colaboradores do departamento ao qual pertencem.

Solução:

```
1 SELECT CODIGOFUNCIONARIO, NOME, SALARIO
2 FROM FUNCIONARIO F
3 WHERE SALARIO>
4                 (SELECT AVG(SALARIO)
5                 FROM FUNCIONARIO
6                 WHERE CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

Fonte: O autor.

O resultado da consulta 05 está expresso na figura a seguir:

	codigofuncionario	nome	salario
1	1	ROBERTA SILVA BRASIL	7.000
2	2	MARIA SILVA BRASIL	9.500

Fonte: O autor.

Resultado da consulta 05.

TRATA-SE DE UMA CONSULTA COM LÓGICA DE CONSTRUÇÃO SEMELHANTE AO QUE FOI DESENVOLVIDO NA CONSULTA 02. NO ENTANTO, A MÉDIA SALARIAL É CALCULADA LEVANDO EM CONSIDERAÇÃO SOMENTE OS FUNCIONÁRIOS DE CADA DEPARTAMENTO. ISSO OCORRE EM FUNÇÃO DA CONDIÇÃO WHERE DECLARADA NA LINHA 6.

Há outra maneira de resolver a consulta 05?

Sim, você pode trocar uma subconsulta por uma junção. O código a seguir produz os mesmos resultados:

```sql
SELECT CODIGOFUNCIONARIO, NOME, SALARIO
FROM FUNCIONARIO F
     JOIN
                (SELECT CODIGODEPARTAMENTO,AVG(SALARIO) AS MEDIA
                FROM FUNCIONARIO
                GROUP  BY CODIGODEPARTAMENTO) TESTE
    ON F.CODIGODEPARTAMENTO=TESTE.CODIGODEPARTAMENTO
WHERE SALARIO>MEDIA;
```

Fonte: O autor.

O exemplo a seguir mostra o uso de uma consulta correlacionada em uma operação de atualização (UPDATE) nos dados.

Consulta 06: suponha que surgiu a necessidade de equiparar os salários dos funcionários que atuam no mesmo departamento. Os funcionários de cada departamento terão salário atualizado em função do maior salário dos seus setores.

Observe na figura a seguir os salários “antes” da atualização.

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	6.000	1
3	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	6.000	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Fonte: O autor.

📷 Tabela FUNCIONARIO antes da atualização dos salários.

Perceba que a listagem está ordenada pela coluna CODIGODEPARTAMENTO. O maior salário de funcionário pertencente ao departamento 1 é R$ 7.000; em relação ao departamento 2, R$ 9.500. Note também que há um funcionário sem a informação sobre departamento.

Solução:

```sql
UPDATE FUNCIONARIO F
SET SALARIO=
                (SELECT MAX(SALARIO)
                FROM FUNCIONARIO
                WHERE CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO)
WHERE F.CODIGODEPARTAMENTO IS NOT NULL;
```

Fonte: O autor.

Observe na figura a seguir os salários após a atualização salarial.

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario	codigodepartamento
1	1	ROBERTA SILVA BRASIL	[NULL]	F	1980-02-20	7.000	1
2	3	GABRIELLA PEREIRA LIMA	[NULL]	F	1990-02-20	7.000	1
3	2	MARIA SILVA BRASIL	[NULL]	F	1988-09-20	9.500	2
4	4	MARCOS PEREIRA BRASIL	[NULL]	M	1999-02-20	9.500	2
5	5	HEMERSON SILVA BRASIL	[NULL]	M	1992-12-20	4.000	[NULL]

Fonte: O autor.

Tabela FUNCIONARIO depois da atualização dos salários.

TRATA-SE DE UMA CONSULTA COM OBJETIVO DE RECUPERAR O MAIOR SALÁRIO DENTRO DO CONTEXTO DE CADA DEPARTAMENTO E, EM SEGUIDA, USAR ESSE VALOR PARA ATUALIZAÇÃO SALARIAL NA TABELA FUNCIONARIO, DE ACORDO COM O DEPARTAMENTO DE CADA COLABORADOR. PERCEBA TAMBÉM QUE A ATUALIZAÇÃO OCORRE SOMENTE PARA OS FUNCIONÁRIOS PARA OS QUAIS EXISTE ALOCAÇÃO A DEPARTAMENTO, OU SEJA, A CLÁUSULA WHERE DA LINHA 6 INIBE ATUALIZAÇÃO DE SALÁRIO CASO NÃO HAJA DEPARTAMENTO ASSOCIADO A ALGUM COLABORADOR.

A sintaxe completa do comando UPDATE no PostgreSQL A sintaxe completa do comando UPDATE pode ser encontrada com a indicação disponibilizada no Explore + ao final do tema.

CONSULTA CORRELACIONADA COM USO DE [NOT] EXISTS

Podemos utilizar o operador EXISTS em uma consulta correlacionada. Tal operador testa a existência de linha(s) retornada(s) por alguma subconsulta. Veja o exemplo a seguir:

Consulta 07: exibir o código e o nome do departamento onde há pelo menos um funcionário alocado.

Solução:

```
1 SELECT D.CODIGODEPARTAMENTO, D.NOME
2 FROM DEPARTAMENTO D
3 WHERE EXISTS
4       (SELECT F.CODIGODEPARTAMENTO
5       FROM FUNCIONARIO F
6       WHERE D.CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

Fonte: O autor.

Observe na figura a seguir os resultados da consulta.

	codigodepartamento	nome
1	1	Tecnologia da Informação
2	2	Contabilidade

Fonte: O autor.

📷 Departamentos onde há pelo menos um funcionário alocado.

💬 A SUBCONSULTA CORRELACIONADA (LINHAS 4 A 6) É EXECUTADA. CASO HAJA PELO MENOS UMA LINHA EM SEU RETORNO, A AVALIAÇÃO DA CLÁUSULA WHERE RETORNA VERDADEIRO E AS COLUNAS ESPECIFICADAS NA LINHA 1 SÃO EXIBIDAS.

Finalmente, se estivéssemos interessados em saber se há departamento sem ocorrência de colaborador alocado, bastaria usar a negação (NOT), conforme a seguir:

```sql
SELECT D.CODIGODEPARTAMENTO, D.NOME
FROM DEPARTAMENTO D
WHERE NOT EXISTS
      (SELECT F.CODIGODEPARTAMENTO
      FROM FUNCIONARIO F
      WHERE D.CODIGODEPARTAMENTO=F.CODIGODEPARTAMENTO);
```

Fonte: O autor.

Observe na figura a seguir os resultados da consulta.

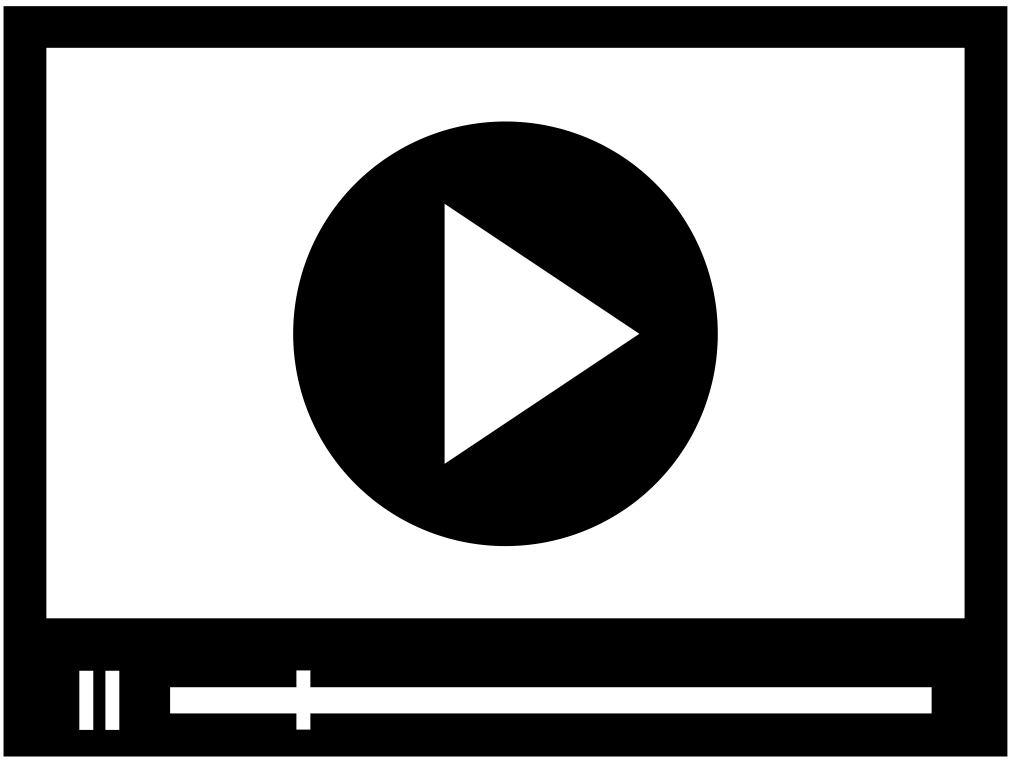
	codigodepartamento	nome
1	3	Marketing

Fonte: O autor.

📷 Departamento onde não há funcionário alocado.

Durante o estudo deste módulo, aprendemos a desenvolver consultas que dependem dos resultados de outras consultas para exibir de maneira correta os seus resultados aos usuários. Tais consultas são denominadas aninhadas.

Ainda, conhecemos a estrutura de consultas correlacionadas, vistas como um tipo especial de consulta aninhada, na qual cada linha da consulta externa é testada junto à consulta interna.



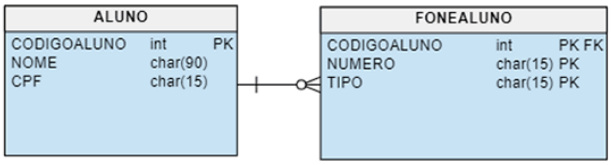
CONSULTAS NO POSTGRESQL SOBRE AS TABELAS FUNCIONÁRIO E DEPARTAMENTO

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Fonte: Shutterstock.

VERIFICANDO O APRENDIZADO

1. CONSIDERE AS TABELAS E O CÓDIGO SQL A SEGUIR:




```sql
SELECT CODIGOALUNO, NOME
FROM ALUNO
WHERE EXISTS
        (SELECT CODIGOALUNO
        FROM FONEALUNO
        WHERE FONEALUNO.CODIGOALUNO=ALUNO.CODIGOALUNO);
```

A CONSULTA RETORNA:

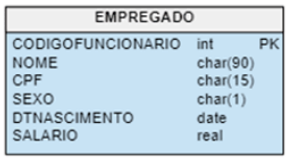
A) Somente o(s) registro(s) de aluno(s) sem telefone.

B) O(s) registro(s) de aluno(s) com pelo menos um telefone.

C) Somente o(s) registro(s) de aluno(s) com pelo menos dois telefones.

D) Somente o(s) registro(s) de aluno(s) com pelo menos três telefones.

2. CONSIDERE A TABELA E O CÓDIGO SQL A SEGUIR:




```sql
SELECT COUNT(*) AS QUANTIDADE
FROM EMPREGADO
WHERE SALARIO<
        (SELECT MAX(SALARIO)
        FROM EMPREGADO );
```

A CONSULTA RETORNA:

A) O número de empregados que ganham menos que o empregado de maior salário.

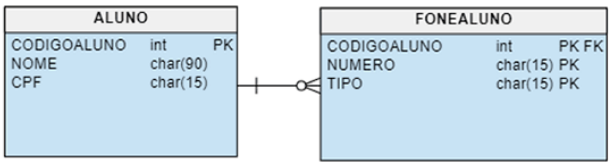
B) O número de empregados que ganham menos que a média salarial das colaboradoras.

C) O maior salário entre as colaboradoras.

D) O número de funcionárias.

GABARITO

1. Considere as tabelas e o código SQL a seguir:



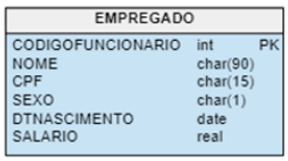

```
1 SELECT CODIGOALUNO, NOME
2 FROM ALUNO
3 WHERE EXISTS
4         (SELECT CODIGOALUNO
5         FROM FONEALUNO
6         WHERE FONEALUNO.CODIGOALUNO=ALUNO.CODIGOALUNO);
```

A consulta retorna:

A alternativa **"B "** está correta.

Há o uso de uma subconsulta correlata, junto a cláusula EXISTS, que testa a existência de algum retorno da subconsulta. Assim, caso haja retorno da consulta, significa que o aluno em questão possui pelo menos um telefone.

2. Considere a tabela e o código SQL a seguir:




```
1 SELECT COUNT(*) AS QUANTIDADE
2 FROM EMPREGADO
3 WHERE SALARIO<
4         (SELECT MAX(SALARIO)
5         FROM EMPREGADO );
```

A consulta retorna:

A alternativa **"A "** está correta.

A cláusula WHERE compara o salário de cada funcionário junto ao do colaborador que tem maior renda. Caso o salário do funcionário seja menor que o maior salário, esse funcionário é considerado na contagem.

MÓDULO 3

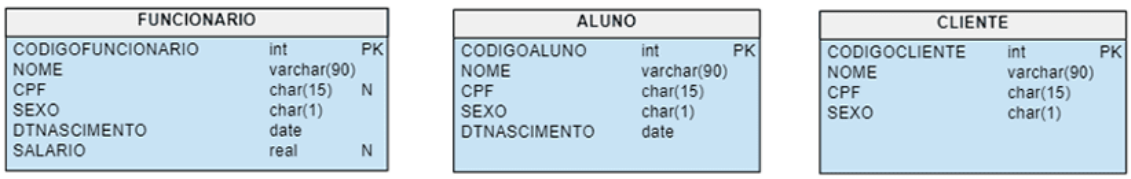
- Operar consultas com o uso de operadores de conjunto

OPERADORES DE CONJUNTO

Vamos aprender que os resultados de diversas consultas podem ser combinados em um único conjunto de dados, caso sigam regras específicas dos operadores utilizados para essa finalidade. Estamos falando dos operadores de conjunto, que incluem UNION, INTERSECT e EXCEPT.

Exemplo de tabelas

Construiremos as consultas com base nas tabelas FUNCIONARIO, ALUNO e CLIENTE, conforme figura a seguir:



Fonte: O autor.

Tabelas FUNCIONARIO, ALUNO e CLIENTE.

Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o *script* a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e acessar algum *database* criado por você.

```sql
CREATE TABLE FUNCIONARIO (
   CODIGOFUNCIONARIO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CPF char(15)  NULL,
   SEXO char(1)  NOT NULL,
   DTNASCIMENTO date  NOT NULL,
   SALARIO real   NULL,
   CONSTRAINT FUNCIONARIO_pk PRIMARY KEY (CODIGOFUNCIONARIO));
CREATE TABLE ALUNO (
   CODIGOALUNO int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CPF char(15) NOT NULL,
   SEXO char(1)  NOT NULL,
   DTNASCIMENTO date  NOT NULL,
   CONSTRAINT ALUNO_pk PRIMARY KEY (CODIGOALUNO));
 CREATE TABLE CLIENTE (
   CODIGOCLIENTE int  NOT NULL,
   NOME varchar(90)  NOT NULL,
   CPF char(15)  NOT NULL,
   SEXO char(1)  NOT NULL,
   CONSTRAINT CLIENTE_pk PRIMARY KEY (CODIGOCLIENTE));
```

Fonte: O autor.

Criação das tabelas FUNCIONARIO, ALUNO e CLIENTE.

O *script* a seguir pode ser utilizado para inserção de registros nas tabelas.

```sql
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (1,'ROBERTA SILVA BRASIL','82998','F','20/02/1980',7000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (2,'MARIA SILVA BRASIL','9876','F','20/09/1988',9500);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (3,'GABRIELLA PEREIRA LIMA','32998','F','20/02/1990',6000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (4,'MARCOS PEREIRA BRASIL','9999','M','20/02/1999',6000);
INSERT INTO FUNCIONARIO (CODIGOFUNCIONARIO, NOME, CPF, SEXO, DTNASCIMENTO, SALARIO)
VALUES (5,'HEMERSON SILVA BRASIL','9111','M','20/12/1992',4000);

INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES   (1,'JOSÉ FRANCISCO TERRA','82988','M','28/10/1989');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES   (2,'ANDREY COSTA FILHO','0024','M','20/10/1999');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES   (3,'ROBERTA SILVA BRASIL','82998','F','20/02/1980');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES   (4,'CARLA MARIA MACIEL','0044','F','20/11/1996');
INSERT INTO ALUNO (CODIGOALUNO, NOME, CPF, SEXO, DTNASCIMENTO) VALUES   (5,'MARCOS PEREIRA BRASIL','9999','M','20/02/1999');

INSERT INTO CLIENTE (CODIGOCLIENTE, NOME, CPF, SEXO) VALUES   (1,'ROBERTA SILVA BRASIL','82998','F');
INSERT INTO CLIENTE (CODIGOCLIENTE, NOME, CPF, SEXO) VALUES   (2,'MARCOS PEREIRA BRASIL','9999','M');
INSERT INTO CLIENTE (CODIGOCLIENTE, NOME, CPF, SEXO) VALUES   (3,'HEMERSON SILVA BRASIL','9111','M');
```

Fonte: O autor.

Inserção de dados nas tabelas FUNCIONARIO, ALUNO e CLIENTE.

A figura a seguir apresenta o conteúdo da tabela FUNCIONARIO após a execução do comando SELECT * FROM FUNCIONARIO;

	codigofuncionario	nome	cpf	sexo	dtnascimento	salario
1	1	ROBERTA SILVA BRASIL	82998	F	1980-02-20	7.000
2	2	MARIA SILVA BRASIL	9876	F	1988-09-20	9.500
3	3	GABRIELLA PEREIRA LIMA	32998	F	1990-02-20	6.000
4	4	MARCOS PEREIRA BRASIL	9999	M	1999-02-20	6.000
5	5	HEMERSON SILVA BRASIL	9111	M	1992-12-20	4.000

Fonte: O autor.

Conteúdo da tabela FUNCIONARIO.

A figura a seguir apresenta o conteúdo da tabela ALUNO após a execução do comando TABLE ALUNO;

	codigoaluno	nome	cpf	sexo	dtnascimento
1	1	JOSÉ FRANCISCO TERRA	82988	M	1989-10-28
2	2	ANDREY COSTA FILHO	0024	M	1999-10-20
3	3	ROBERTA SILVA BRASIL	82998	F	1980-02-20
4	4	CARLA MARIA MACIEL	0044	F	1996-11-20
5	5	MARCOS PEREIRA BRASIL	9999	M	1999-02-20

Fonte: O autor.

Conteúdo da tabela ALUNO.

A figura a seguir apresenta o conteúdo da tabela CLIENTE após a execução do comando TABLE CLIENTE;

	codigocliente	nome	cpf	sexo
1	1	ROBERTA SILVA BRASIL	82998	F
2	2	MARCOS PEREIRA BRASIL	9999	M
3	3	HEMERSON SILVA BRASIL	9111	M

Fonte: O autor.

Conteúdo da tabela CLIENTE.

Convém esclarecer que, em todos os registros, os dados são puramente fictícios. Além disso, os dados da coluna CPF estão com somente quatro caracteres e não representam um CPF válido. No entanto, vamos considerar, para efeitos do nosso estudo, que registros com mesmo valor de CPF representam a informação de um mesmo cidadão. De forma complementar, estamos levando em consideração que cada tabela pertence a um banco de dados – e um domínio de aplicação – diferente.

CONSULTAS COM O OPERADOR UNION

O operador de união serve para consolidar linhas resultantes de consultas. Para isso, todas as consultas envolvidas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipo de dados. Além disso, linhas repetidas são eliminadas do resultado,

uma vez que o resultado é uma tabela que não permite duplicata de linhas. O operador de união possui a seguinte forma geral:

CONSULTASQL UNION [ALL|DISTINCT] CONSULTASQL

A sintaxe completa do comando SELECT no PostgreSQL pode ser pode ser encontrada com a indicação disponibilizada no Explore + ao final do tema.

Consulta 01: Retornar o nome e o CPF de todos os funcionários e clientes.

Solução:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 UNION
4 SELECT NOME, CPF
5 FROM CLIENTE;
```

Fonte: O autor.

O resultado da consulta 01 está expresso na figura a seguir:

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	GABRIELLA PEREIRA LIMA	32998
3	MARCOS PEREIRA BRASIL	9999
4	MARIA SILVA BRASIL	9876
5	HEMERSON SILVA BRASIL	9111

Fonte: O autor.

Resultado da consulta 01.

COMENTÁRIO

Perceba que há cinco funcionários cadastrados (linhas 1 a 10 do script de inserção) e, de forma semelhante, três clientes (linhas 18 a 20 do script de inserção). Note também que todos os clientes cadastrados também são funcionários. Após o processamento da operação de união, somente cinco registros foram exibidos, uma vez que as repetições por padrão são eliminadas.

E se quiséssemos que todos registros aparecessem no resultado?

Bastaria usar o UNION ALL, conforme a seguir:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 UNION ALL
4 SELECT NOME, CPF
5 FROM CLIENTE;
```

Fonte: O autor.

O resultado da consulta 01 modificada está expresso na figura a seguir:

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	MARIA SILVA BRASIL	9876
3	GABRIELLA PEREIRA LIMA	32998
4	MARCOS PEREIRA BRASIL	9999
5	HEMERSON SILVA BRASIL	9111
6	ROBERTA SILVA BRASIL	82998
7	MARCOS PEREIRA BRASIL	9999
8	HEMERSON SILVA BRASIL	9111

Fonte: O autor.

📷 Resultado da consulta 01 modificada para contemplar todos os registros das tabelas.

Finalmente, se quiséssemos especificar a “origem” de cada registro, poderíamos alterar o nosso código conforme a seguir:

```
1 SELECT NOME, CPF, 'Dados da tabela FUNCIONARIO' AS ORIGEM
2 FROM FUNCIONARIO
3 UNION ALL
4 SELECT NOME, CPF, 'Dados da tabela CLIENTE' AS ORIGEM
5 FROM CLIENTE;
```

Fonte: O autor.

O resultado da consulta 01 modificada está expresso na figura a seguir:

	nome	cpf	origem
1	ROBERTA SILVA BRASIL	82998	Dados da tabela FUNCIONARIO
2	MARIA SILVA BRASIL	9876	Dados da tabela FUNCIONARIO
3	GABRIELLA PEREIRA LIMA	32998	Dados da tabela FUNCIONARIO
4	MARCOS PEREIRA BRASIL	9999	Dados da tabela FUNCIONARIO
5	HEMERSON SILVA BRASIL	9111	Dados da tabela FUNCIONARIO
6	ROBERTA SILVA BRASIL	82998	Dados da tabela CLIENTE
7	MARCOS PEREIRA BRASIL	9999	Dados da tabela CLIENTE
8	HEMERSON SILVA BRASIL	9111	Dados da tabela CLIENTE

Fonte: O autor.

📷 Resultado da consulta 01 modificada para contemplar todos os registros das tabelas exibir a tabela de origem.

CONSULTAS COM O OPERADOR INTERSECT

O operador de interseção serve para exibir linhas que aparecem em ambos os resultados das consultas envolvidas. Para isso, todas as consultas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipo de dados. Além disso, linhas repetidas são eliminadas do resultado. O operador de interseção possui a seguinte forma geral:

CONSULTASQL INTERSECT [ALL|DISTINCT] CONSULTASQL

Vamos estudar alguns exemplos?

Consulta 02: Retornar o nome e o CPF de todos os cidadãos que são funcionários e clientes.

Solução:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 INTERSECT
4 SELECT NOME, CPF
5 FROM CLIENTE;
```

Fonte: O autor.

O resultado da consulta 02 está expresso na figura a seguir:

	nome	cpf
1	MARCOS PEREIRA BRASIL	9999
2	ROBERTA SILVA BRASIL	82998
3	HEMERSON SILVA BRASIL	9111

Fonte: O autor.

Resultado da consulta 02.

A consulta retorna três linhas que são fruto da interseção entre as tabelas FUNCIONARIO e CLIENTE. Como visto, todos os clientes são funcionários.

Consulta 03: Retornar o nome e o CPF de todos os cidadãos que são funcionários, clientes e alunos.

Solução:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 INTERSECT
4 SELECT NOME, CPF
5 FROM CLIENTE
6 INTERSECT
7 SELECT NOME, CPF
8 FROM ALUNO;
```

Fonte: O autor.

O resultado da consulta 03 está expresso na figura a seguir:

	nome	cpf
1	MARCOS PEREIRA BRASIL	9999
2	ROBERTA SILVA BRASIL	82998

Fonte: O autor.

Resultado da consulta 03.

A consulta retorna duas linhas que são fruto da interseção entre as tabelas FUNCIONARIO, CLIENTE e ALUNO.

Um aspecto importante é que uma consulta sob o formato X UNION Y INTERSECT Z é interpretada sendo X UNION (Y INTERSECT Z). Veja o exemplo a seguir:

```sql
SELECT NOME, CPF
FROM FUNCIONARIO
UNION
SELECT NOME, CPF
FROM CLIENTE
INTERSECT
SELECT NOME, CPF
FROM ALUNO;
```

Fonte: O autor.

O resultado da consulta envolvendo as três tabelas está expresso na figura a seguir:

	nome	cpf
1	ROBERTA SILVA BRASIL	82998
2	GABRIELLA PEREIRA LIMA	32998
3	MARCOS PEREIRA BRASIL	9999
4	MARIA SILVA BRASIL	9876
5	HEMERSON SILVA BRASIL	9111

Fonte: O autor.

📷 Resultado da consulta envolvendo três tabelas.

CONSULTAS COM O OPERADOR EXCEPT

O operador EXCEPT implementa a operação de subtração da Teoria dos Conjuntos e serve para exibir linhas que aparecem em uma consulta e não aparecem na outra. Para isso, todas as consultas devem possuir a mesma quantidade de colunas e deve haver compatibilidade de tipo de dados. Além disso, linhas repetidas são eliminadas do resultado. O operador de subtração possui a seguinte forma geral:

CONSULTASQL EXCEPT [ALL|DISTINCT] CONSULTASQL

Convém ressaltar que alguns SGBDs implementam a mesmo operador, usando um nome diferente. O Oracle, por exemplo, utiliza o operador MINUS, significando subtração ou diferença.

Vamos estudar alguns exemplos?

Consulta 04: Retornar o nome e o CPF dos funcionários que não são clientes.

Solução:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 EXCEPT
4 SELECT NOME, CPF
5 FROM CLIENTE;
```

Fonte: O autor.

O resultado da consulta 04 está expresso na figura a seguir:

	nome	cpf
1	GABRIELLA PEREIRA LIMA	32998
2	MARIA SILVA BRASIL	9876

Fonte: O autor.

Resultado da consulta 04.

A consulta retorna duas linhas que são fruto da subtração entre as tabelas FUNCIONARIO e CLIENTE.

Perceba que uma operação X EXCEPT Y é diferente de Y EXCEPT X. Veja o código a seguir:

```
1 SELECT NOME, CPF
2 FROM CLIENTE
3 EXCEPT
4 SELECT NOME, CPF
5 FROM FUNCIONARIO;
```

Fonte: O autor.

A consulta retorna vazio pois todos os clientes são funcionários.

Consulta 05: Retornar o nome e o CPF dos cidadãos que são somente funcionários.

Solução:

```
1 SELECT NOME, CPF
2 FROM FUNCIONARIO
3 EXCEPT
4 SELECT NOME, CPF
5 FROM CLIENTE
6 EXCEPT
7 SELECT NOME, CPF
8 FROM ALUNO;
```

Fonte: O autor.

O resultado da consulta 05 está expresso na figura a seguir:

	nome	cpf
1	GABRIELLA PEREIRA LIMA	32998
2	MARIA SILVA BRASIL	9876

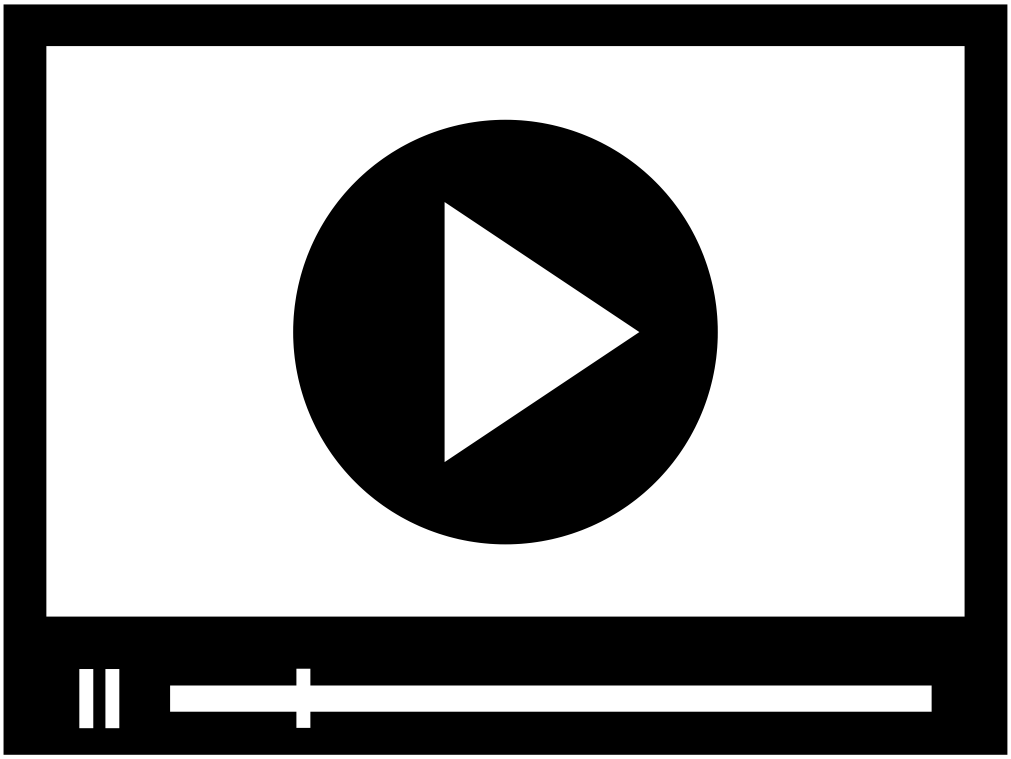
Fonte: O autor.

Resultado da consulta 05.

COMENTÁRIO

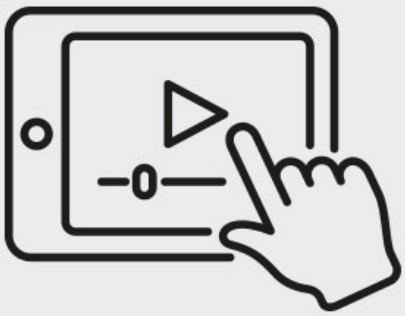
Inicialmente, o SGBD processa a operação de subtração da linha 3. Em seguida, o resultado da operação é usado na subtração da linha 6.

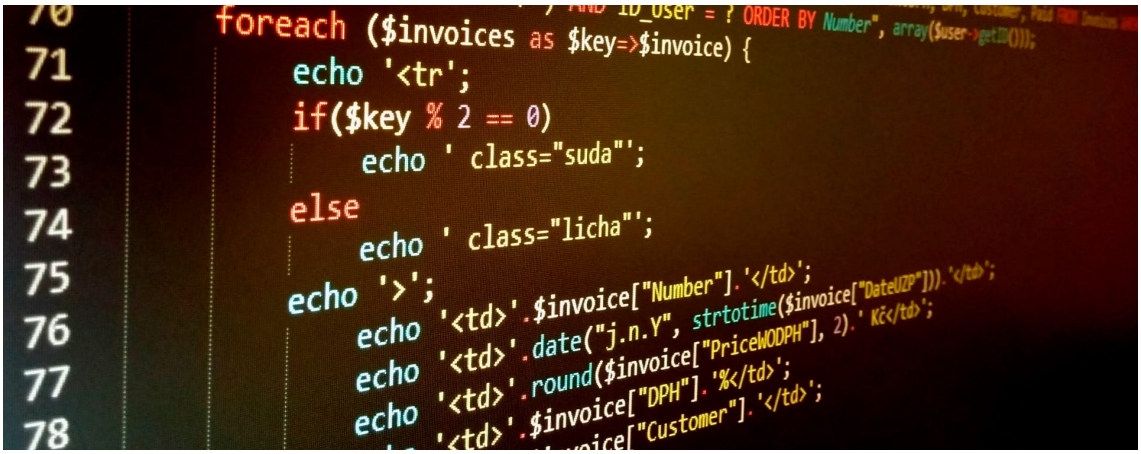
Ao longo deste módulo, aprendemos a construir consultas envolvendo operadores de conjunto. Foram analisados os operadores UNION, INTERSECT e EXCEPT. Todos têm em comum a obrigatoriedade de serem usadas consultas com o mesmo número de colunas, além da compatibilidade entre os tipos de dados delas.



CONSULTAS NO POSTGRESQL SOBRE AS TABELAS FUNCIONÁRIO, ALUNO E CLIENTE

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

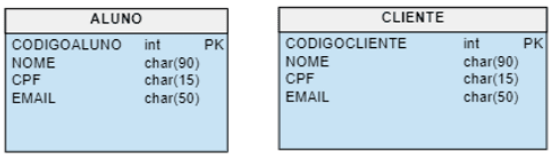




Fonte: Shutterstock.

VERIFICANDO O APRENDIZADO

1. CONSIDERE QUE CADA TABELA A SEGUIR PERTENCE A UM BANCO DE DADOS DISTINTO.



QUAL CONSULTA RETORNA O NOME E O CPF DOS ALUNOS QUE NÃO SÃO CLIENTES?

A) SELECT NOME, CPF FROM ALUNO

EXCEPT

SELECT NOME, CPF FROM CLIENTE;

B) SELECT NOME, CPF FROM ALUNO

UNION

SELECT NOME, CPF FROM CLIENTE;

C) SELECT NOME, CPF FROM ALUNO

UNION ALL

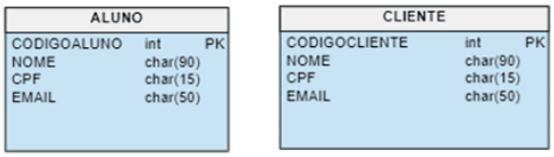
SELECT NOME, CPF FROM CLIENTE;

D) SELECT NOME, CPF FROM ALUNO

INTERSECT

SELECT NOME, CPF FROM CLIENTE;

2. CONSIDERE QUE CADA TABELA A SEGUIR PERTENCE A UM BANCO DE DADOS DISTINTO.



QUAL CONSULTA RETORNA O NOME E O CPF DOS CLIENTES QUE SÃO ALUNOS?

A) SELECT NOME, CPF FROM ALUNO

EXCEPT

SELECT NOME, CPF FROM CLIENTE;

B) SELECT NOME, CPF FROM CLIENTE

INTERSECT

SELECT NOME, CPF FROM ALUNO;

C) SELECT NOME, CPF FROM ALUNO

UNION ALL

SELECT NOME, CPF FROM CLIENTE;

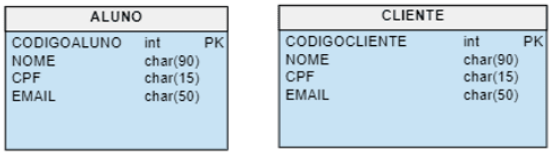
D) SELECT NOME, CPF FROM CLIENTE

JOIN

SELECT NOME, CPF FROM ALUNO;

GABARITO

1. Considere que cada tabela a seguir pertence a um banco de dados distinto.

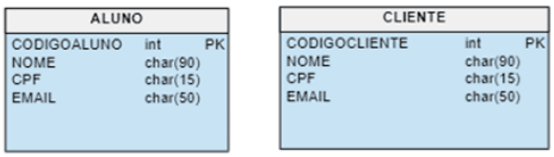


Qual consulta retorna o nome e o CPF dos alunos que não são clientes?

A alternativa **"A "** está correta.

Para recuperar os alunos que não são clientes, basta utilizar o operador de subtração, que, no contexto da questão, retornará os registros resultantes da consulta envolvendo a tabela ALUNO que não estão presentes nos resultados da consulta envolvendo a tabela CLIENTE.

2. Considere que cada tabela a seguir pertence a um banco de dados distinto.



Qual consulta retorna o nome e o CPF dos clientes que são alunos?

A alternativa **"B "** está correta.

Para recuperar os clientes que são alunos, basta utilizar o operador de interseção, que, no contexto da questão, retornará os registros resultantes da consulta envolvendo a tabela CLIENTE que não estão presentes nos resultados da consulta envolvendo a tabela ALUNO.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Neste tema, vimos que as consultas SQL envolvem, em geral, mais de uma tabela. Nessas consultas, é essencial a operação de junção entre as tabelas, que tem dois tipos: junção interna (INNER JOIN) e externa (OUTER JOIN), dependendo do resultado que se deseja obter da consulta.

Vimos, também, que as consultas em SQL podem ser aninhadas, em decorrência da propriedade de que o resultado de uma consulta é uma tabela e, por isso, pode estar sucessivamente sujeito a novas consultas. Dentre as consultas aninhadas, vimos um tipo especial denominado de consulta correlata.

Finalmente, aplicamos as operações de união, interseção e diferença da Teoria Matemática dos Conjuntos nas consultas SQL, através dos operadores UNION, INTERSECT e EXCEPT.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.

AVALIAÇÃO DO TEMA

REFERÊNCIAS

POSTGRESQL. **Manual de referência do PostgreSQL**. *IN*: POSTGRESQL. Consultado em meio eletrônico em: 11 Ago 2020.

EXPLORE+

Para saber mais sobre os comandos CREATE, ALTER E INSERT explorados neste tema, acesse:

- Postgresql em create table; altertable; sql-altertable.
- Postgresql em sql-select, e entenda o comando INSERT.
- Postgresql em sql-uptade, e entenda o comando UPDATE.

CONTEUDISTA

Nathielly de Souza Campos

🔗 CURRÍCULO LATTES